UNIs: sg3637, um2147

Task 1

In [210]:

```python
import pandas as pd
import warnings
from IPython.display import display
warnings.filterwarnings('ignore')
pd.options.display.max_columns = None
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import numpy as np
warnings.simplefilter('ignore')
```

In [4]:

```python
#Reading Sample from the Research Data Table
import random

filename = "data/OP_DTL_RSRCH_PGYR2017_P01182019.csv"
n = sum(1 for line in open(filename)) - 1
#number of records in file (excludes header)
s = 100000
#desired sample size
skip = sorted(random.sample(range(1,n+1),n-s))
#the 0-indexed header will not be included in the skip list
dfResearch = pd.read_csv(filename, skiprows=skip)

dfResearch.shape
```

Out[4]:

```
(100000, 176)
```

In [5]:

```
dfResearch.head()
```

Out[5]:

Recipient_Primary_Business_Street_Address_Line2	Recipient_City	Recipient_State	Recipient_Zi
NaN	Fridley	MN	
Suite 255	San Diego	CA	
NaN	KALAMAZOO	MI	
NaN	LAWRENCEVILLE	NJ	
NaN	JEFFERSONVILLE	IN	

In [6]:

```
#Viewing all the Columns from dfResearch
dfResearch.columns
```

Out[6]:

```
Index(['Change_Type', 'Covered_Recipient_Type',
       'Noncovered_Recipient_Entity_Name', 'Teaching_Hospital_CCN',
       'Teaching_Hospital_ID', 'Teaching_Hospital_Name',
       'Physician_Profile_ID', 'Physician_First_Name', 'Physician_Midd
le_Name',
       'Physician_Last_Name',
       ...
       'Preclinical_Research_Indicator', 'Delay_in_Publication_Indicat
or',
       'Name_of_Study', 'Dispute_Status_for_Publication', 'Record_ID',
       'Program_Year', 'Payment_Publication_Date',
       'ClinicalTrials_Gov_Identifier', 'Research_Information_Link',
       'Context_of_Research'],
      dtype='object', length=176)
```

In [7]:

```
#As per the data dictionary, "Physicians may be identified as covered recipients
# with research-related payment records. Hence, we will  replace the missing val

dfResearch['Physician_Primary_Type'].fillna(dfResearch['Principal_Investigator_1_
dfResearch['Physician_Specialty'].fillna(dfResearch['Principal_Investigator_1_Sp
dfResearch['Physician_License_State_code1'].fillna(dfResearch['Principal_Investi
```

In [8]:

```
#Read from Chunk with condition
#https://stackoverflow.com/questions/34549402/pandas-read-a-small-random-sample
```

In [9]:

```python
#Reading Sample for the other table
import random

filename = "data/OP_DTL_GNRL_PGYR2017_P01182019.csv"
n = sum(1 for line in open(filename)) - 1
#number of records in file (excludes header)
s = 100000
#desired sample size
skip = sorted(random.sample(range(1,n+1),n-s))
#the 0-indexed header will not be included in the skip list
df = pd.read_csv(filename, skiprows=skip)
```

In [10]:

```python
#df consists of payments made in general/ not via research. Adding the label: I
df['Label'] = 0
dfResearch['Label'] = 1
```

In [34]:

```python
#Combining the two dataframes in df_final
df_final = pd.concat([dfResearch, df], join="inner")
df_final = df_final.sample(frac=1, random_state=42).reset_index(drop=True)

#Dropping columns with all na values
df_final.dropna(axis=1, how='all',inplace = True)
```

Identifying Features

In the initial feature selection, we will use heuristics to reduce the of features that evidently would not impact the model. We will then use more statistical methods and visulization to further select on the relevant features.

1. We will drop all the features that have greater than 50% null values.

In [35]:

```python
#Dropping Features that have >50% null values
df_final = df_final.loc[:, df_final.isnull().mean() <= .5]
```

In [36]:

```
df_final.head()
```

Out[36]:

	Change_Type	Covered_Recipient_Type	Physician_Profile_ID	Physician_First_Name	Physician_L
0	UNCHANGED	Covered Recipient Physician	3297.0	JOHN	
1	UNCHANGED	Non-covered Recipient Entity	NaN	NaN	
2	UNCHANGED	Covered Recipient Physician	180292.0	KEVIN	
3	UNCHANGED	Non-covered Recipient Entity	NaN	NaN	
4	UNCHANGED	Non-covered Recipient Entity	NaN	NaN	

```
In [37]:
```

```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 33 columns):
Change_Type                                                         20
0000 non-null object
Covered_Recipient_Type                                              20
0000 non-null object
Physician_Profile_ID                                                10
3722 non-null float64
Physician_First_Name                                                10
3721 non-null object
Physician_Last_Name                                                 10
3721 non-null object
Recipient_Primary_Business_Street_Address_Line1                     19
9879 non-null object
Recipient_City                                                      19
9879 non-null object
Recipient_State                                                     19
9799 non-null object
Recipient_Zip_Code                                                  19
9799 non-null object
Recipient_Country                                                   19
9879 non-null object
Physician_Primary_Type                                              19
8794 non-null object
Physician_Specialty                                                 19
8526 non-null object
Physician_License_State_code1                                       19
8791 non-null object
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name           20
0000 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_ID         20
0000 non-null int64
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name       20
0000 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State      18
8624 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country    20
0000 non-null object
Related_Product_Indicator                                           20
0000 non-null object
Covered_or_Noncovered_Indicator_1                                   18
1534 non-null object
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1           16
3439 non-null object
Product_Category_or_Therapeutic_Area_1                              15
8380 non-null object
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1            16
2667 non-null object
Associated_Drug_or_Biological_NDC_1                                 12
1608 non-null object
Total_Amount_of_Payment_USDollars                                   20
0000 non-null float64
Date_of_Payment                                                     20
0000 non-null object
Form_of_Payment_or_Transfer_of_Value                                20
0000 non-null object
```

```
Delay_in_Publication_Indicator                                      20
0000 non-null object
Dispute_Status_for_Publication                                      20
0000 non-null object
Record_ID                                                           20
0000 non-null int64
Program_Year                                                        20
0000 non-null int64
Payment_Publication_Date                                            20
0000 non-null object
Label                                                               20
0000 non-null int64
dtypes: float64(2), int64(4), object(27)
memory usage: 50.4+ MB
```

2. We will remove columns that are evidently dependent on each other. For instance, physician_id will be correlated with Physician_First_Name and Physician_Last_Name. In this case, we will remove Physician_First_Name and Physician_Last_Name. Similar approach has been taken for some of the other features.

In [38]:

```
#Removing some obvious identifier/name columns that can lead to information lea
df_final.drop(columns=['Physician_First_Name', 'Physician_Last_Name', 'Record_II
```

In [39]:

```
df_final.head()
```

Out[39]:

	Change_Type	Covered_Recipient_Type	Physician_Profile_ID	Recipient_Primary_Business_Street_
0	UNCHANGED	Covered Recipient Physician	3297.0	2525 W
1	UNCHANGED	Non-covered Recipient Entity	NaN	530 NE
2	UNCHANGED	Covered Recipient Physician	180292.0	1435 N Ra
3	UNCHANGED	Non-covered Recipient Entity	NaN	12
4	UNCHANGED	Non-covered Recipient Entity	NaN	2141 E

In [40]:

```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 30 columns):
Change_Type                                                     20
0000 non-null object
Covered_Recipient_Type                                          20
0000 non-null object
Physician_Profile_ID                                            10
3722 non-null float64
Recipient_Primary_Business_Street_Address_Line1                 19
9879 non-null object
Recipient_City                                                  19
9879 non-null object
Recipient_State                                                 19
9799 non-null object
Recipient_Zip_Code                                              19
9799 non-null object
Recipient_Country                                               19
9879 non-null object
Physician_Primary_Type                                          19
8794 non-null object
Physician_Specialty                                             19
8526 non-null object
Physician_License_State_code1                                   19
8791 non-null object
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name       20
0000 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_ID     20
0000 non-null int64
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name   20
0000 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State  18
8624 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country 20
0000 non-null object
Related_Product_Indicator                                       20
0000 non-null object
Covered_or_Noncovered_Indicator_1                               18
1534 non-null object
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1       16
3439 non-null object
Product_Category_or_Therapeutic_Area_1                          15
8380 non-null object
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1        16
2667 non-null object
Associated_Drug_or_Biological_NDC_1                             12
1608 non-null object
Total_Amount_of_Payment_USDollars                               20
0000 non-null float64
Date_of_Payment                                                 20
0000 non-null object
Form_of_Payment_or_Transfer_of_Value                            20
0000 non-null object
Delay_in_Publication_Indicator                                  20
0000 non-null object
Dispute_Status_for_Publication                                  20
0000 non-null object
```

```
Program_Year                                                        20
0000 non-null int64
Payment_Publication_Date                                            20
0000 non-null object
Label                                                               20
0000 non-null int64
dtypes: float64(2), int64(3), object(25)
memory usage: 45.8+ MB
```

Let us know look at the number of unique values in each of the columns. For columns with categorical variables and less than 60 unique values, we will visualize the number of data points in each of the categories through bar charts. This will allow us to examine any categories that are there in only one class. We will eliminate such records as this will allow us to minimize data leakage.

In [41]:

```
df_final.nunique().sort_values()
```

Out[41]:

```
Program_Year
1
Delay_in_Publication_Indicator
1
Payment_Publication_Date
1
Label
2
Dispute_Status_for_Publication
2
Covered_or_Noncovered_Indicator_1
2
Related_Product_Indicator
2
Change_Type
3
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1
4
Covered_Recipient_Type
4
Form_of_Payment_or_Transfer_of_Value
4
Physician_Primary_Type
6
Recipient_Country
7
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country
26
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State
43
Physician_License_State_code1
54
Recipient_State
56
Physician_Specialty
275
Date_of_Payment
365
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name
874
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_ID
979
Associated_Drug_or_Biological_NDC_1
993
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name
993
Product_Category_or_Therapeutic_Area_1                              1
060
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1            3
941
Recipient_City                                                      8
122
Recipient_Zip_Code                                                 38
028
Total_Amount_of_Payment_USDollars                                  46
```

```
399
Recipient_Primary_Business_Street_Address_Line1                        73
954
Physician_Profile_ID                                                   73
964
dtype: int64
```

In [42]:

```
#Dropping the columns that have only 1 unique value = Zero Variance
df_final.drop(columns=['Program_Year', 'Delay_in_Publication_Indicator', 'Paymen
```

In [43]:

```
df_final.nunique().sort_values()
```

Out[43]:

```
Label
2
Covered_or_Noncovered_Indicator_1
2
Related_Product_Indicator
2
Dispute_Status_for_Publication
2
Change_Type
3
Form_of_Payment_or_Transfer_of_Value
4
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1
4
Covered_Recipient_Type
4
Physician_Primary_Type
6
Recipient_Country
7
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country
26
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State
43
Physician_License_State_code1
54
Recipient_State
56
Physician_Specialty
275
Date_of_Payment
365
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name
874
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_ID
979
Associated_Drug_or_Biological_NDC_1
993
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name
993
Product_Category_or_Therapeutic_Area_1                               1
060
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1             3
941
Recipient_City                                                       8
122
Recipient_Zip_Code                                                  38
028
Total_Amount_of_Payment_USDollars                                   46
399
Recipient_Primary_Business_Street_Address_Line1                     73
954
Physician_Profile_ID                                                73
964
dtype: int64
```

In [44]:

```
df_final.head()
```

Out[44]:

	Change_Type	Covered_Recipient_Type	Physician_Profile_ID	Recipient_Primary_Business_Street_
0	UNCHANGED	Covered Recipient Physician	3297.0	2525 W
1	UNCHANGED	Non-covered Recipient Entity	NaN	530 NE
2	UNCHANGED	Covered Recipient Physician	180292.0	1435 N Ra
3	UNCHANGED	Non-covered Recipient Entity	NaN	12
4	UNCHANGED	Non-covered Recipient Entity	NaN	2141 E

In [45]:

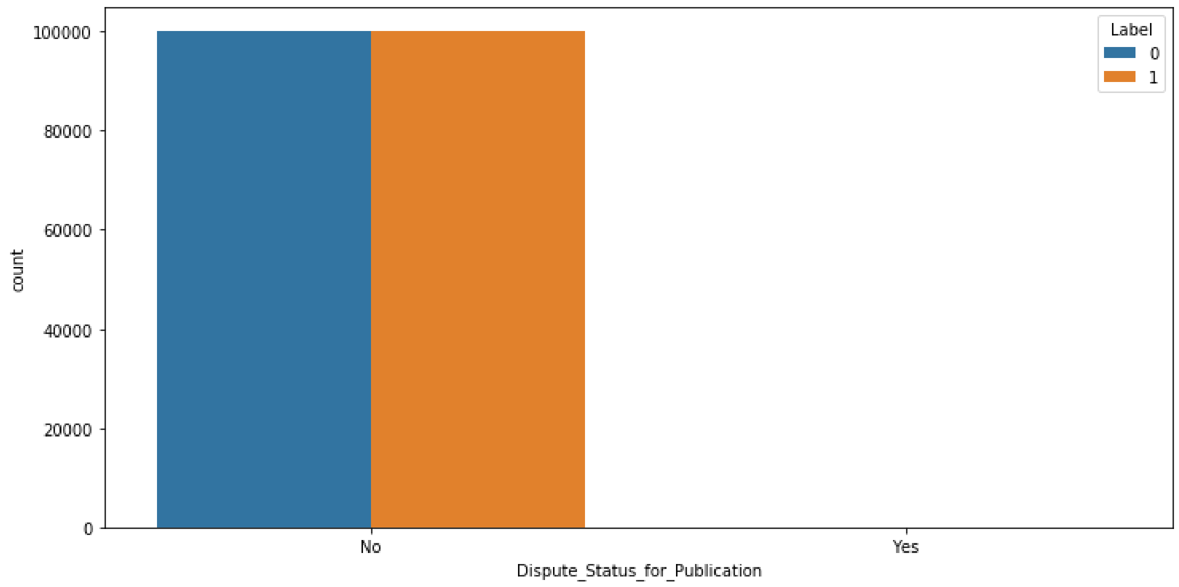
```
#Dispute_Status_for_Publication
df_final.groupby(['Label', 'Dispute_Status_for_Publication']).count()
```

Out[45]:

		Change_Type	Covered_Recipient_Type	Physician_Profile_ID
Label	**Dispute_Status_for_Publication**			
0	**No**	99989	99989	99610
	Yes	11	11	8
1	**No**	99955	99955	4097
	Yes	45	45	7

In [46]:

```
plt.figure(figsize  = (12,6))
_ = sns.countplot(x="Dispute_Status_for_Publication", data=df_final, hue='Label
```



We will keep this column as it does not lead to information leakage. Both classes have 'Yes' and 'No'.

In [47]:

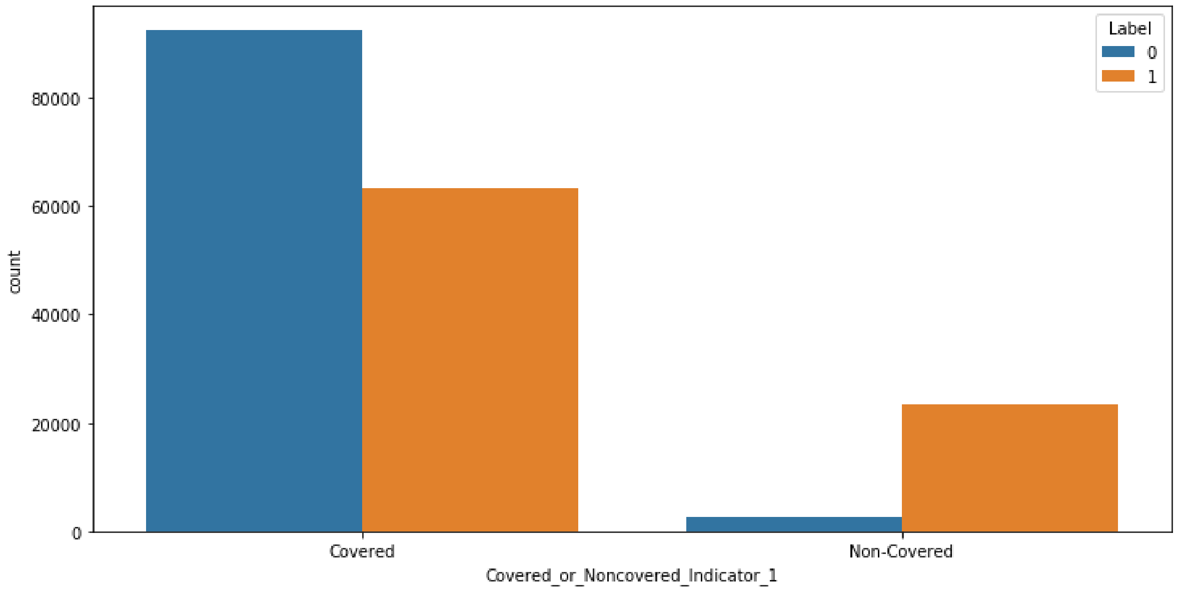
```
#Covered_or_Noncovered_Indicator_1
df_final.groupby(['Label', 'Covered_or_Noncovered_Indicator_1']).count()
```

Out[47]:

		Change_Type	Covered_Recipient_Type	Physician_Prof
Label	**Covered_or_Noncovered_Indicator_1**			
0	**Covered**	92343	92343	
	Non-Covered	2648	2648	
1	**Covered**	63205	63205	
	Non-Covered	23338	23338	

In [48]:

```
plt.figure(figsize  = (12,6))
_ = sns.countplot(x="Covered_or_Noncovered_Indicator_1", data=df_final, hue='Lal
```



We will keep this column as it does not lead to information leakage. Both classes have 'Covered' and 'Non-Covered'

In [49]:

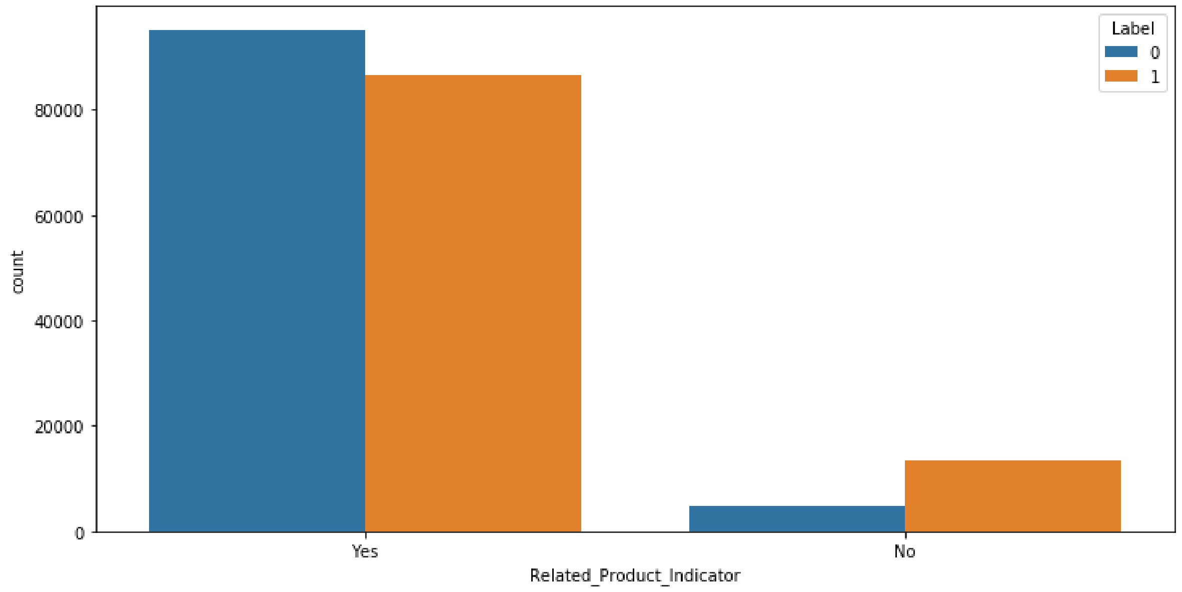
```
#Related_Product_Indicator
df_final.groupby(['Label', 'Related_Product_Indicator']).count()
```

Out[49]:

		Change_Type	Covered_Recipient_Type	Physician_Profile_ID	Re
Label	**Related_Product_Indicator**				
0	**No**	5009	5009	4915	
	Yes	94991	94991	94703	
1	**No**	13457	13457	503	
	Yes	86543	86543	3601	

In [50]:

```
plt.figure(figsize  = (12,6))
_ = sns.countplot(x="Related_Product_Indicator", data=df_final, hue='Label')
```



We will keep this column as it does not lead to information leakage. Both classes have 'Yes' and 'No'

In [51]:

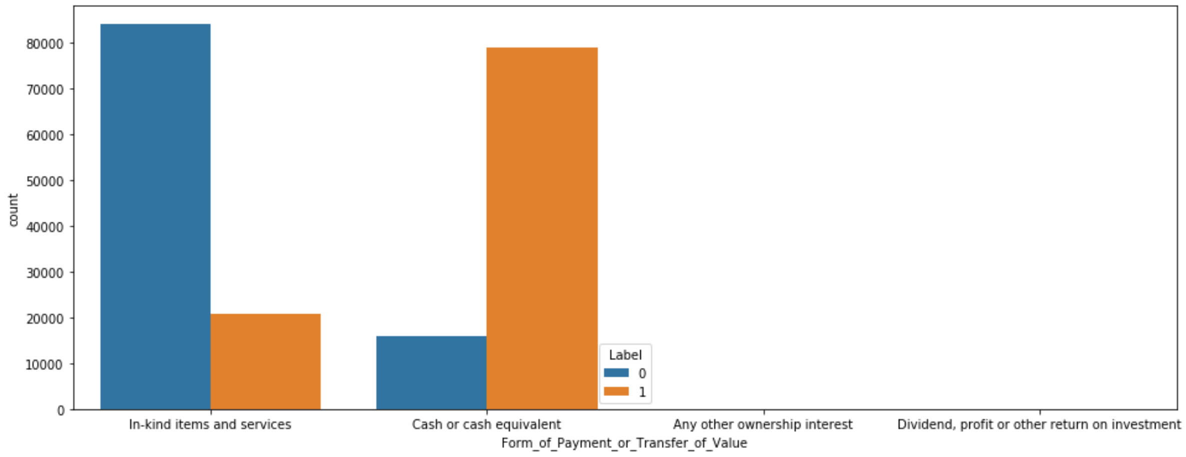
```
#Form_of_Payment_or_Transfer_of_Value
df_final.groupby(['Label', 'Form_of_Payment_or_Transfer_of_Value']).count()
```

Out[51]:

		Change_Type	Covered_Recipient_Type	Physician_P
Label	**Form_of_Payment_or_Transfer_of_Value**			
0	**Any other ownership interest**	1	1	
	Cash or cash equivalent	15882	15882	
	Dividend, profit or other return on investment	4	4	
	In-kind items and services	84113	84113	
1	**Cash or cash equivalent**	79129	79129	
	In-kind items and services	20871	20871	

In [52]:

```
plt.figure(figsize  = (16,6))
_ = sns.countplot(x="Form_of_Payment_or_Transfer_of_Value", data=df_final, hue=
```



From the above chart, it looks like 'Form_of_Payment_or_Transfer_of_Value' is very much correlated with the target label. As such, we will drop this feature completely.

In [53]:

```
df_final.drop(columns='Form_of_Payment_or_Transfer_of_Value', inplace=True)
```

In [54]:

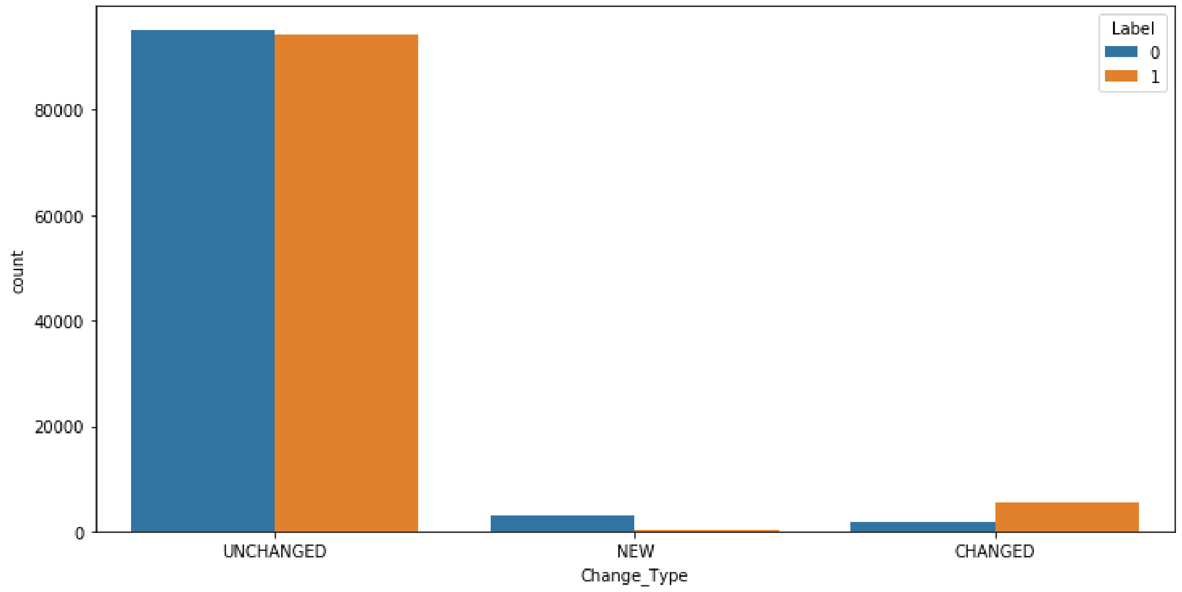
```
#Change_Type
df_final.groupby(['Label', 'Change_Type']).count()
```

Out[54]:

Label	Change_Type	Covered_Recipient_Type	Physician_Profile_ID	Recipient_Primary_Business_St
0	CHANGED	1904	1899	
	NEW	3059	3048	
	UNCHANGED	95037	94671	
1	CHANGED	5470	86	
	NEW	276	123	
	UNCHANGED	94254	3895	

In [55]:

```
plt.figure(figsize  = (12,6))
_ = sns.countplot(x="Change_Type", data=df_final, hue='Label')
```



We will keep this column as it does not lead to information leakage. Both classes have same categories.

In [56]:

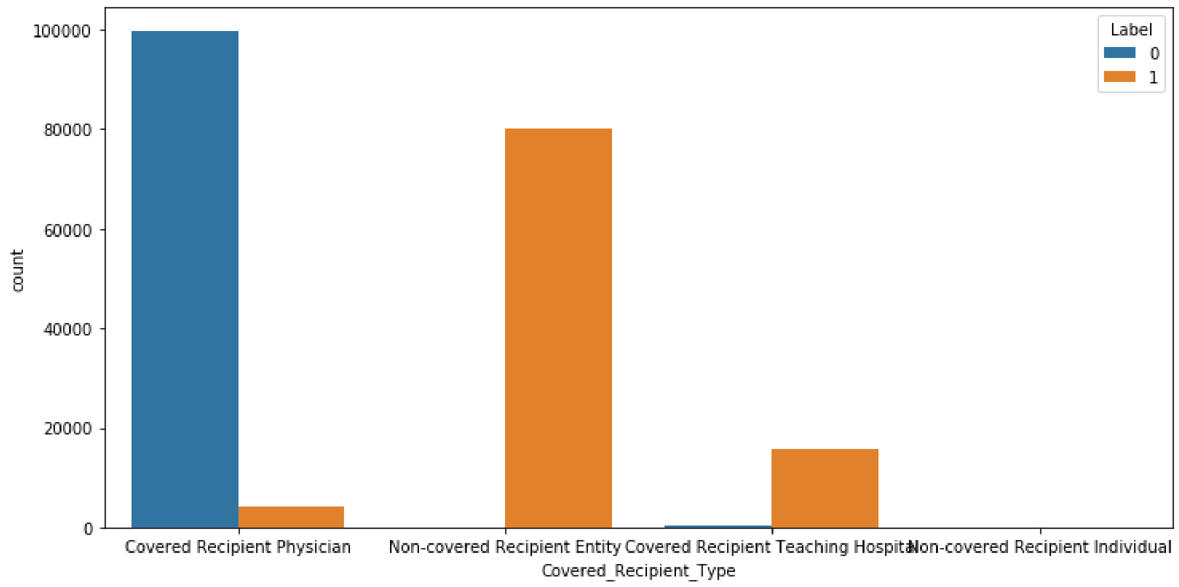
```
#Covered_Recipient_Type
df_final.groupby(['Label', 'Covered_Recipient_Type']).count()
```

Out[56]:

		Change_Type	Physician_Profile_ID	Recipient_Primary_Business_St
Label	**Covered_Recipient_Type**			
0	**Covered Recipient Physician**	99618	99618	
	Covered Recipient Teaching Hospital	382	0	
1	**Covered Recipient Physician**	4104	4104	
	Covered Recipient Teaching Hospital	15771	0	
	Non-covered Recipient Entity	80004	0	
	Non-covered Recipient Individual	121	0	

In [57]:

```python
plt.figure(figsize  = (12,6))
_ = sns.countplot(x="Covered_Recipient_Type", data=df_final, hue='Label')
```



From the above chart, it looks like 'Covered_Recipient_Type' is very much correlated with the target label. As such, we will drop this feature completely.

In [58]:

```python
df_final.drop(columns='Covered_Recipient_Type', inplace=True)
```

In [59]:

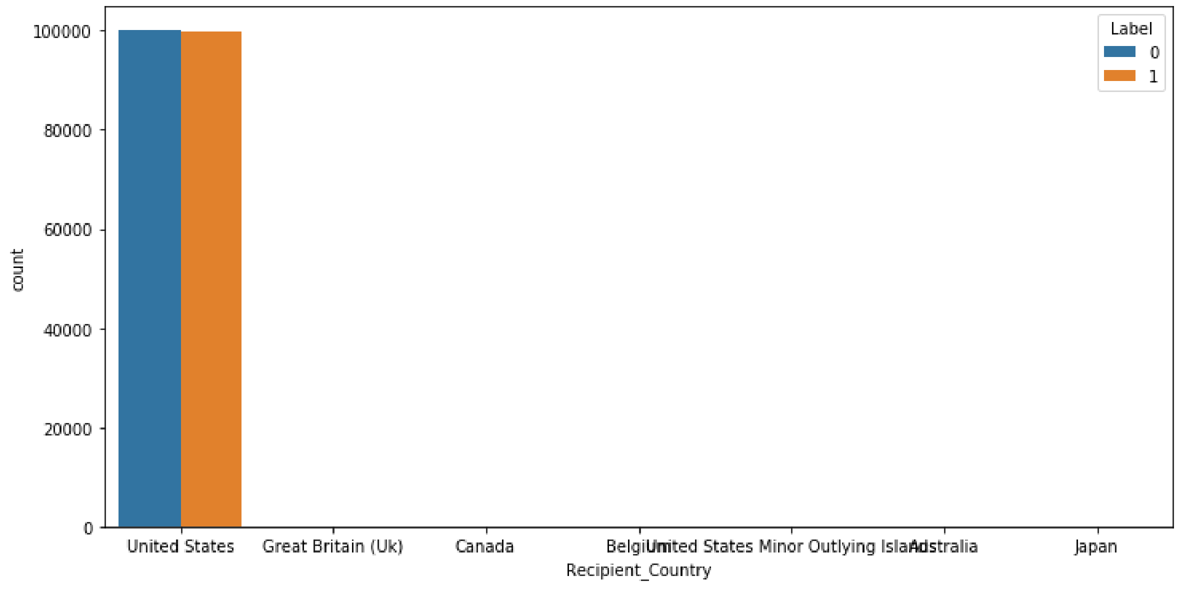
```
#Recipient_Country
df_final.groupby(['Label', 'Recipient_Country']).count()
```

Out[59]:

		Change_Type	Physician_Profile_ID	Recipient_Primary_Business_Street_A
Label	**Recipient_Country**			
0	**Australia**	1	1	
	Great Britain (Uk)	1	1	
	United States	99997	99615	
	United States Minor Outlying Islands	1	1	
1	**Belgium**	1	0	
	Canada	10	0	
	Great Britain (Uk)	64	0	
	Japan	1	0	
	United States	99802	4104	
	United States Minor Outlying Islands	1	0	

In [60]:

```
plt.figure(figsize = (12,6))
_ = sns.countplot(x="Recipient_Country", data=df_final, hue='Label')
```



From the above, we can see that most of the data has Recipient_Country as United States. As such, there is very little variance. In order to avoid data leakage, we will remove this feature all together.

In [61]:

```
#Dropping Recipient_Country Column
df_final.drop(columns='Recipient_Country', inplace=True)
```

In [62]:

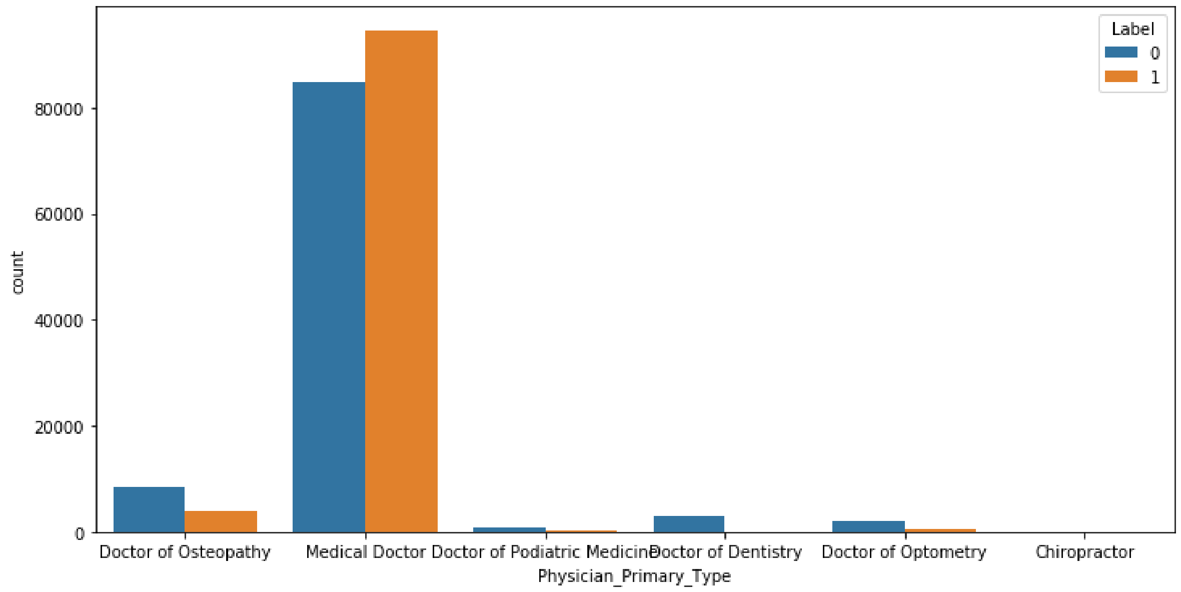
```
#Physician_Primary_Type
df_final.groupby(['Label', 'Physician_Primary_Type']).count()
```

Out[62]:

		Change_Type	Physician_Profile_ID	Recipient_Primary_Business_Stı
Label	**Physician_Primary_Type**			
0	**Chiropractor**	41	41	
	Doctor of Dentistry	3110	3110	
	Doctor of Optometry	2200	2200	
	Doctor of Osteopathy	8535	8535	
	Doctor of Podiatric Medicine	916	916	
	Medical Doctor	84816	84816	
1	**Chiropractor**	1	1	
	Doctor of Dentistry	98	65	
	Doctor of Optometry	459	69	
	Doctor of Osteopathy	3977	145	
	Doctor of Podiatric Medicine	219	8	
	Medical Doctor	94422	3816	

In [63]:

```
plt.figure(figsize  = (12,6))
_ = sns.countplot(x="Physician_Primary_Type", data=df_final, hue='Label')
#Remove this feature due to information leakage
```



From the above we can see that the Chiropractor Physician_Primary_Type is specific to label 0, hence to avoid information leakage we will remove rows where Physician_Primary_Type = Chiropractor

In [64]:

```
df_final = df_final[(df_final.Physician_Primary_Type != 'Chiropractor')]
```

In [65]:

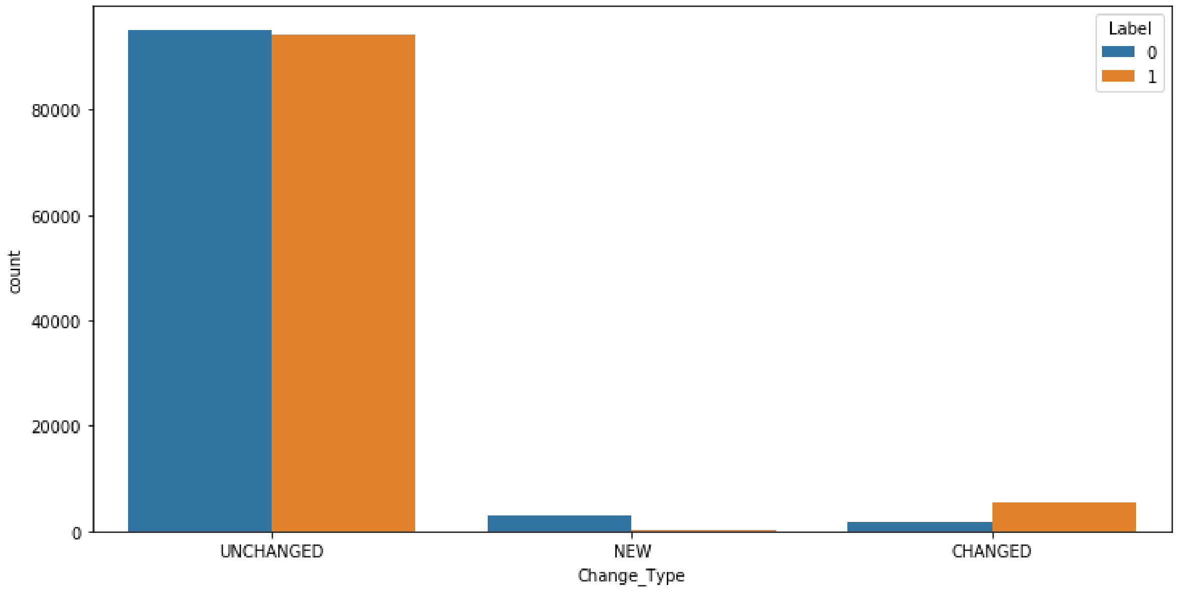
```
#Change_Type
df_final.groupby(['Label', 'Change_Type']).count()
```

Out[65]:

		Physician_Profile_ID	Recipient_Primary_Business_Street_Address_Line1	Reci
Label	**Change_Type**			
0	**CHANGED**	1899	1904	
	NEW	3048	3059	
	UNCHANGED	94630	94996	
1	**CHANGED**	86	5470	
	NEW	123	276	
	UNCHANGED	3894	94132	

In [66]:

```
plt.figure(figsize  = (12,6))
_ = sns.countplot(x="Change_Type", data=df_final, hue='Label')
```



We will keep this column as it does not lead to information leakage. Both classes have same categories.

In [67]:

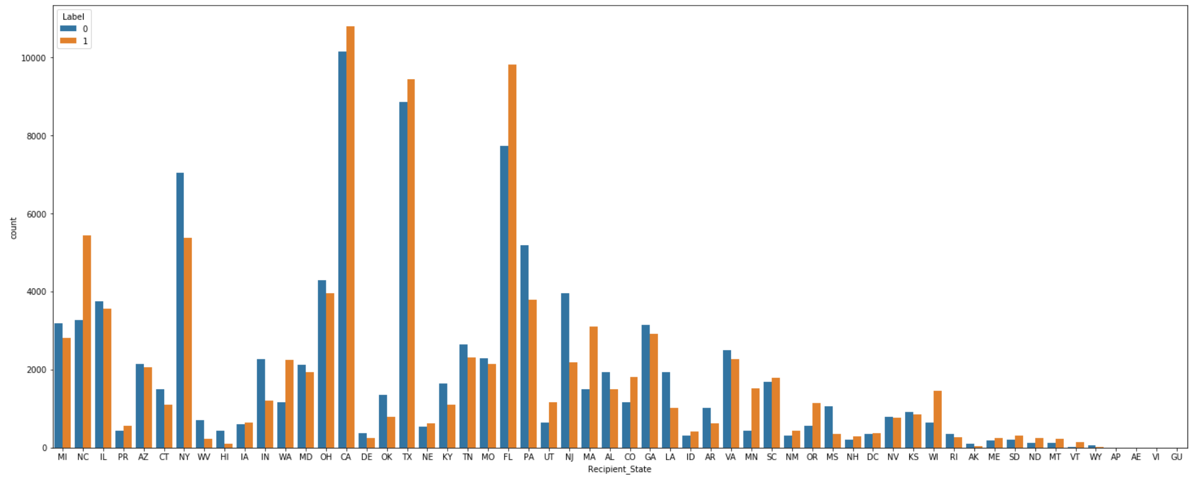
```
#Recipient_State
df_final.groupby(['Label', 'Recipient_State']).count()
```

Out[67]:

		Change_Type	Physician_Profile_ID	Recipient_Primary_Business_Street_Address_Line1	Reci
Label	**Recipient_State**				
0	**AE**	3	3	3	
	AK	101	101	101	
	AL	1941	1932	1941	
	AP	2	2	2	
	AR	1014	1012	1014	
	AZ	2153	2147	2153	
	CA	10144	10098	10144	
	CO	1173	1170	1173	
	CT	1509	1502	1509	

In [68]:

```python
plt.figure(figsize  = (25,10))
_ = sns.countplot(x="Recipient_State", data=df_final, hue='Label')
```



As we can see from the above chart, both classes have presence in each of the states. As such, we will preserve this feature and will not eliminate any rows.

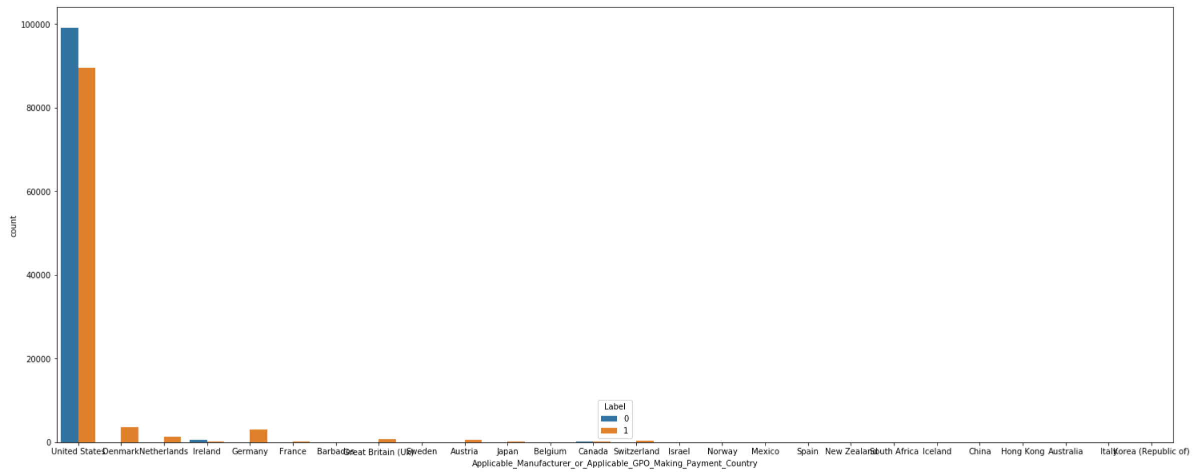
In [69]:

```python
#Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country
df_final.groupby(['Label', 'Applicable_Manufacturer_or_Applicable_GPO_Making_Pa
```

```
1                         Australia        1        0
                            Austria      540        0
                           Barbados       16        0
                            Belgium       29        4
                             Canada      124       14
                            Denmark     3681       14
                             France      219        1
                            Germany     3032       21
                 Great Britain (Uk)      817       37
                            Ireland      135        0
                             Israel        8        0
                              Japan      100        7
                        Netherlands     1242       51
```

In [70]:

```
plt.figure(figsize  = (25,10))
_ = sns.countplot(x="Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Co
```



From the above table and chart, it is difficult to see the difference in category. For this, we will adopt a different approach where we will calculate the set difference to find out the categories that are missing in either of the classes.

In [71]:

```
#Countries not in Class 0
a = list(set(df_final[df_final.Label == 1].Applicable_Manufacturer_or_Applicabl
a
```

Out[71]:

```
['Austria', 'Spain', 'New Zealand', 'Norway']
```

In [72]:

```
#Countries not in Class 1
b = list(set(df_final[df_final.Label == 0].Applicable_Manufacturer_or_Applicabl
b
```

Out[72]:

```
['South Africa',
 'China',
 'Italy',
 'Mexico',
 'Hong Kong',
 'Korea (Republic of)',
 'Iceland']
```

In [73]:

```
#Removing rows with the above countries
df_final = df_final.loc[~df_final.Applicable_Manufacturer_or_Applicable_GPO_Mak
```

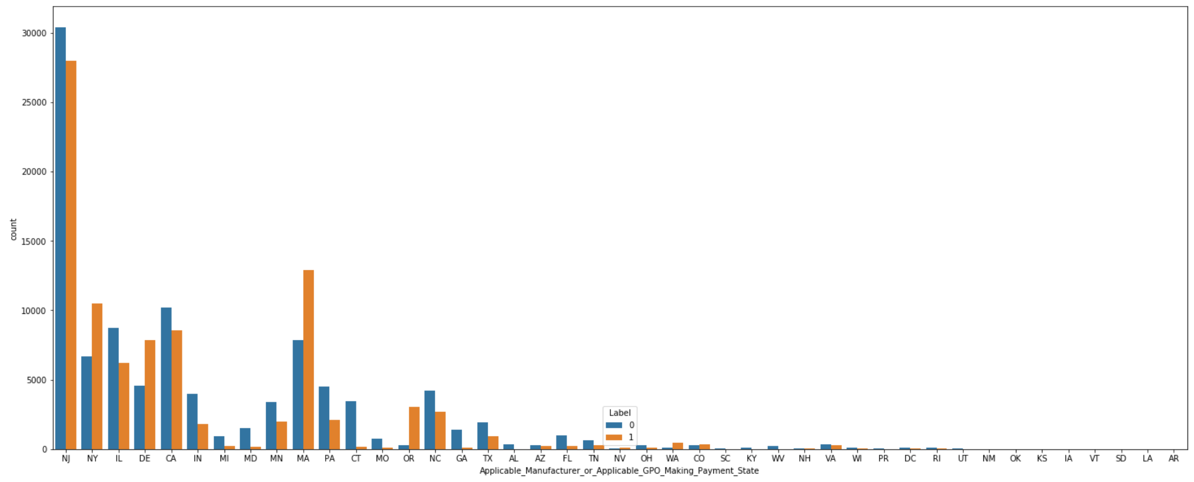
In [74]:

```
#Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State
df_final.groupby(['Label', 'Applicable_Manufacturer_or_Applicable_GPO_Making_Pa
```

NC	2681	389
NH	36	2
NJ	27990	976
NV	116	0
NY	10495	380
OH	105	41
OR	3039	86
PA	2122	57
PR	13	4
RI	63	33
TN	290	0
TX	911	102

In [75]:

```
plt.figure(figsize  = (25,10))
_ = sns.countplot(x="Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_S
```



Similar to the previous feature, it is difficult to see the difference in category. For this, we will adopt a similar approach where we will calculate the set difference to find out the categories that are missing in either of the classes.

In [76]:

```
#Countries not in Class 0
a = list(set(df_final[df_final.Label == 1].Applicable_Manufacturer_or_Applicabl
a
```

Out[76]:

```
['VT']
```

In [77]:

```python
#Countries not in Class 1
b = list(set(df_final[df_final.Label == 0].Applicable_Manufacturer_or_Applicabl
b
```

Out[77]:

```
['KY', 'AR', 'IA', 'SD', 'NM', 'KS', 'LA', 'OK', 'AL', 'SC']
```

In [78]:

```python
#Removing rows with the above countries
df_final = df_final.loc[~df_final.Applicable_Manufacturer_or_Applicable_GPO_Mak
```

In [79]:

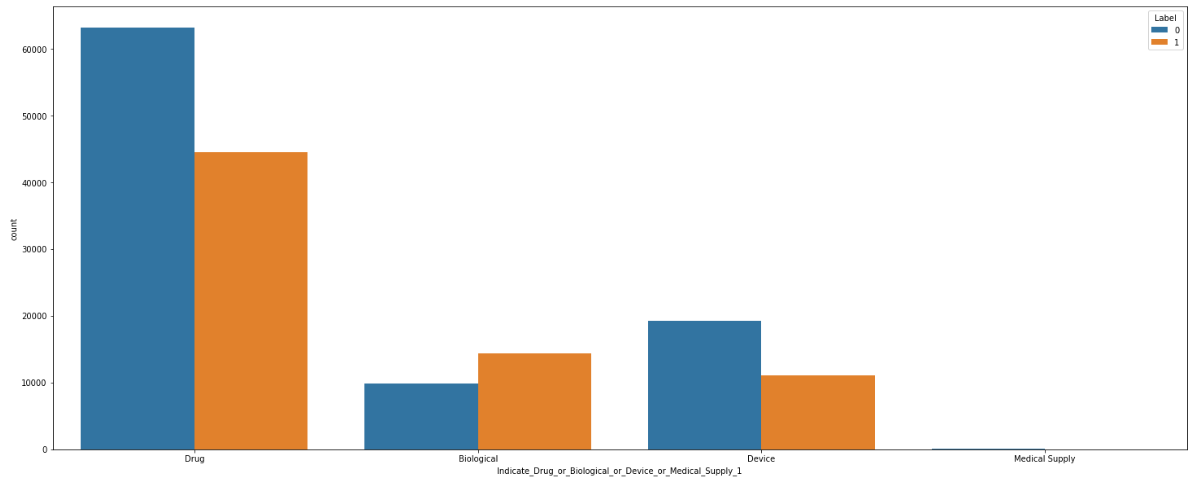
```python
#Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1
df_final.groupby(['Label', 'Indicate_Drug_or_Biological_or_Device_or_Medical_Su
```

Out[79]:

		Change_Type	**Physician_Profile**
Label	**Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1**		
0	**Biological**	9820	97
	Device	19255	19
	Drug	63185	63
	Medical Supply	84	
1	**Biological**	14362	4
	Device	11105	8
	Drug	44553	17
	Medical Supply	13	

In [80]:

```python
plt.figure(figsize  = (25,10))
_ = sns.countplot(x="Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1"
```



As we can see from the above chart, both classes have presence in each of the states. As such, we will preserve this feature and will not eliminate any rows.

In [81]:

```
df_final.drop(columns=['Associated_Drug_or_Biological_NDC_1', 'Physician_Profil
                       'Applicable_Manufacturer_or_Applicable_GPO_Making_Paymen
```

In [82]:

```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 198880 entries, 0 to 199999
Data columns (total 20 columns):
Change_Type                                                        19
8880 non-null object
Recipient_City                                                     19
8759 non-null object
Recipient_State                                                    19
8679 non-null object
Recipient_Zip_Code                                                 19
8679 non-null object
Physician_Primary_Type                                             19
7679 non-null object
Physician_Specialty                                                19
7411 non-null object
Physician_License_State_code1                                      19
7676 non-null object
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name          19
8880 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name      19
8880 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State     18
8072 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country   19
8880 non-null object
Related_Product_Indicator                                          19
8880 non-null object
Covered_or_Noncovered_Indicator_1                                  18
0469 non-null object
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1          16
2377 non-null object
Product_Category_or_Therapeutic_Area_1                             15
7318 non-null object
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1           16
1606 non-null object
Total_Amount_of_Payment_USDollars                                  19
8880 non-null float64
Date_of_Payment                                                    19
8880 non-null object
Dispute_Status_for_Publication                                     19
8880 non-null object
Label                                                              19
8880 non-null int64
dtypes: float64(1), int64(1), object(18)
memory usage: 36.9+ MB
```

In [83]:

```
df_final.head()
```

Out[83]:

	Change_Type	Recipient_City	Recipient_State	Recipient_Zip_Code	Physician_Primary_Type	Ph
0	UNCHANGED	MIDLAND	MI	48642-4600	Doctor of Osteopathy	
1	UNCHANGED	CARY	NC	27518	Doctor of Osteopathy	
2	UNCHANGED	Elgin	IL	60123	Medical Doctor	
3	UNCHANGED	PONCE	PR	00717	Medical Doctor	
4	UNCHANGED	TEMPE	AZ	85282-1892	Doctor of Osteopathy	

In [84]:

```
#Saving the final dataset
df_final.to_csv('data/df_final.csv')
```

Task 2

In the previous part we did intensive identification of the features to prevent information leak. In this section we will build a simple minimum viable model based on the intial features that were selected in previous part.

The final dataset of 200,000 rows that was created will be used here.

In [85]:

```
#Reading the Dataset created in the previous task
data = pd.read_csv('data/df_final.csv')
```

Since, building model on 200,000 is computationally expensive, we will further subsample the dataset to have 20,000 rows.

In [86]:

```
#Creating a sample with only 20% of all the values in the original data.
df_compressed = data.sample(frac= 0.1, random_state=42).reset_index(drop=True)
df_compressed.drop(columns='Unnamed: 0', inplace=True)
df_compressed.groupby(df_compressed['Label']).count()
```

Out[86]:

	Change_Type	Recipient_City	Recipient_State	Recipient_Zip_Code	Physician_Primary_Type
Label					
0	9963	9963	9963	9963	9916
1	9925	9911	9899	9899	9854

As you can see from the above, we have created a balanced dataset through undersampling. Both classes - Class 1 and Class 2 have almost same number of rows in the dataset.

Now, we will preprocess the Date_of_time variable. Since the data is from 2017, we will remove the year and the day and just keep the month, which will be treated as a categorical variable.

In [87]:

```
#Converting the Date_of_Payment to datetime
df_compressed.Date_of_Payment = pd.to_datetime(df_compressed.Date_of_Payment)
#Adding the Month of Payment
df_compressed['Payment_Month'] = df_compressed.Date_of_Payment.dt.month
#Dropping Date_of_Payment (Year is 2017)
df_compressed.drop(columns='Date_of_Payment', inplace=True)
```

In [88]:

```python
df_compressed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19888 entries, 0 to 19887
Data columns (total 20 columns):
Change_Type                                                           19
888 non-null object
Recipient_City                                                        19
874 non-null object
Recipient_State                                                       19
862 non-null object
Recipient_Zip_Code                                                    19
862 non-null object
Physician_Primary_Type                                                19
770 non-null object
Physician_Specialty                                                   19
743 non-null object
Physician_License_State_code1                                         19
770 non-null object
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name             19
888 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name         19
888 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State        18
867 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country      19
888 non-null object
Related_Product_Indicator                                             19
888 non-null object
Covered_or_Noncovered_Indicator_1                                     18
085 non-null object
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1             16
284 non-null object
Product_Category_or_Therapeutic_Area_1                                15
770 non-null object
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1              16
216 non-null object
Total_Amount_of_Payment_USDollars                                     19
888 non-null float64
Dispute_Status_for_Publication                                        19
888 non-null object
Label                                                                 19
888 non-null int64
Payment_Month                                                         19
888 non-null int64
dtypes: float64(1), int64(2), object(17)
memory usage: 3.0+ MB
```

In [89]:

```python
#Converting Payment_Month to object as it is a categorical variable
df_compressed['Payment_Month'] = df_compressed['Payment_Month'].astype(object)
```

In [90]:

```python
X=df_compressed.drop(columns=['Label'])
y=df_compressed.Label
```

We will define the categorical and continuous features. In our dataset, there is only one continuous feature - Total_Amount_of_Payment_USDollars. Rest all the features are categorical.

In [91]:

```
#Categorical Features
cat_features = list(X.columns)
cat_features.remove('Total_Amount_of_Payment_USDollars')

#Continuous Features
cont_features = ['Total_Amount_of_Payment_USDollars']
```

The NaN values for the categorical features will be encoded into a new category 'NA'. There are no null values for the continuous feature.

In [92]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19888 entries, 0 to 19887
Data columns (total 19 columns):
Change_Type                                                           19
888 non-null object
Recipient_City                                                        19
874 non-null object
Recipient_State                                                       19
862 non-null object
Recipient_Zip_Code                                                    19
862 non-null object
Physician_Primary_Type                                                19
770 non-null object
Physician_Specialty                                                   19
743 non-null object
Physician_License_State_code1                                         19
770 non-null object
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name             19
888 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name         19
888 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State        18
867 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country      19
888 non-null object
Related_Product_Indicator                                             19
888 non-null object
Covered_or_Noncovered_Indicator_1                                     18
085 non-null object
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1             16
284 non-null object
Product_Category_or_Therapeutic_Area_1                                15
770 non-null object
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1              16
216 non-null object
Total_Amount_of_Payment_USDollars                                     19
888 non-null float64
Dispute_Status_for_Publication                                        19
888 non-null object
Payment_Month                                                         19
888 non-null object
dtypes: float64(1), object(18)
memory usage: 2.9+ MB
```

In [93]:

```
# Splitting the data in training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

Proceeding to building the baseline model - Logistic Regression.

- We will use limited featured while building the model. As such only use a subset of columns which includes randomly selected 8 features. Columns that would be included are 'Change_Type', 'Recipient_Zip_Code', 'Physician_Specialty', 'Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country', 'Related_Product_Indicator', 'Total_Amount_of_Payment_USDollars', 'Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1', 'Covered_or_Noncovered_Indicator_1'

- Note that above features are chosen on a random basis as this is a baseline model.
- In terms of preprocessing, a new category would be created to encode the null values in each of the column.
- Since this is the baseline model, we will use the default parameters specified in scikit learn for Logistic Regression.

The NaN values for the categorical features will be encoded into a new category 'NA'. There are no null values for the continuous feature.

In [101]:

```
# Replacing the NaN values to NA which is a new category created
def null_categories(Xa):
    Xa = Xa[['Change_Type', 'Recipient_Zip_Code', 'Physician_Specialty', 'Applicable_
             'Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1', 'Covered_or
    return Xa.replace({np.nan:'NA'})
```

In [102]:

```
# Build a pipeline for dealing with categorical variables and continuous variab
from sklearn.compose import make_column_transformer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder

from sklearn.pipeline import make_pipeline

preprocess_baseline_nan = ColumnTransformer(
    [('func', FunctionTransformer(func=null_categories, validate=False), cat_fe

preprocess_baseline_ohe = ColumnTransformer([('ohe', OneHotEncoder(handle_unkno
                                             np.arange(0,8))], remainder='passth
```

In [103]:

```
#Model Building - Baseline Model (Logistic Regression)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

pipeline_baseline = make_pipeline(preprocess_baseline_nan, preprocess_baseline_
logreg_scores = cross_val_score(pipeline_baseline, X_train, y_train, cv=5)
print('Logistic Regression - Accuracy (Baseline Model): ', np.mean(logreg_score
```

```
Logistic Regression - Accuracy (Baseline Model):  0.8429211620271678
```

In [104]:

```
precision_baseline =  cross_val_score(pipeline_baseline, X_train, y_train, cv=5
recall_baseline = cross_val_score(pipeline_baseline, X_train, y_train, cv=5, sc
print('Logistic Regression - Precision (Baseline Model): ', np.mean(precision_b
print('Logistic Regression - Recall (Baseline Model): ', np.mean(recall_baselin
```

```
Logistic Regression - Precision (Baseline Model):  0.873843083096775
Logistic Regression - Recall (Baseline Model):  0.8009128772287095
```

As we can see from the above analysis, a baseline linear model that selects 8 random features achieves an accuracy of 86%. In the subsequent parts, we will try to beat this accuracy using better feature selection mechanisms.

Task 3

In this section, we will create derived features, do more in-depth processing and data cleaning. This will particularly revolve around encoding the categorical variables in different ways. Using this we will check whether this improves the model.

We will follow the following approaches to preprocessing:

- All null values will be encoding in a different category, similar to Task 2.
- Target Encoding will be used to encode categorical features.
- Standardize the continuous feature 'Total_Amount_of_Payment_USDollars'.
- All the features would be used in model building.

In [105]:

```
X.nunique().sort_values()
```

Out[105]:

```
Covered_or_Noncovered_Indicator_1
2
Related_Product_Indicator
2
Dispute_Status_for_Publication
2
Change_Type
3
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1
4
Physician_Primary_Type
5
Payment_Month
12
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country
14
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State
32
Recipient_State
52
Physician_License_State_code1
52
Physician_Specialty                                                  1
90
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name            5
06
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name        5
72
Product_Category_or_Therapeutic_Area_1                               6
29
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1            17
16
Recipient_City                                                      34
80
Recipient_Zip_Code                                                  87
21
Total_Amount_of_Payment_USDollars                                   94
08
dtype: int64
```

In this section, we will drop the obvious correlation features:

- Recipient_City & Recipient_Zip_Code: Correlated with Recipient_State
- Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Country: Correlated with Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State

In [106]:

```
#Dropping redundant features which are obviously correlated
X.drop(columns=['Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Count
               'Recipient_Zip_Code', 'Recipient_City'], inplace=True)

#Finding the Categorical Features
cat_features = list(X.columns)
cat_features.remove('Total_Amount_of_Payment_USDollars')
```

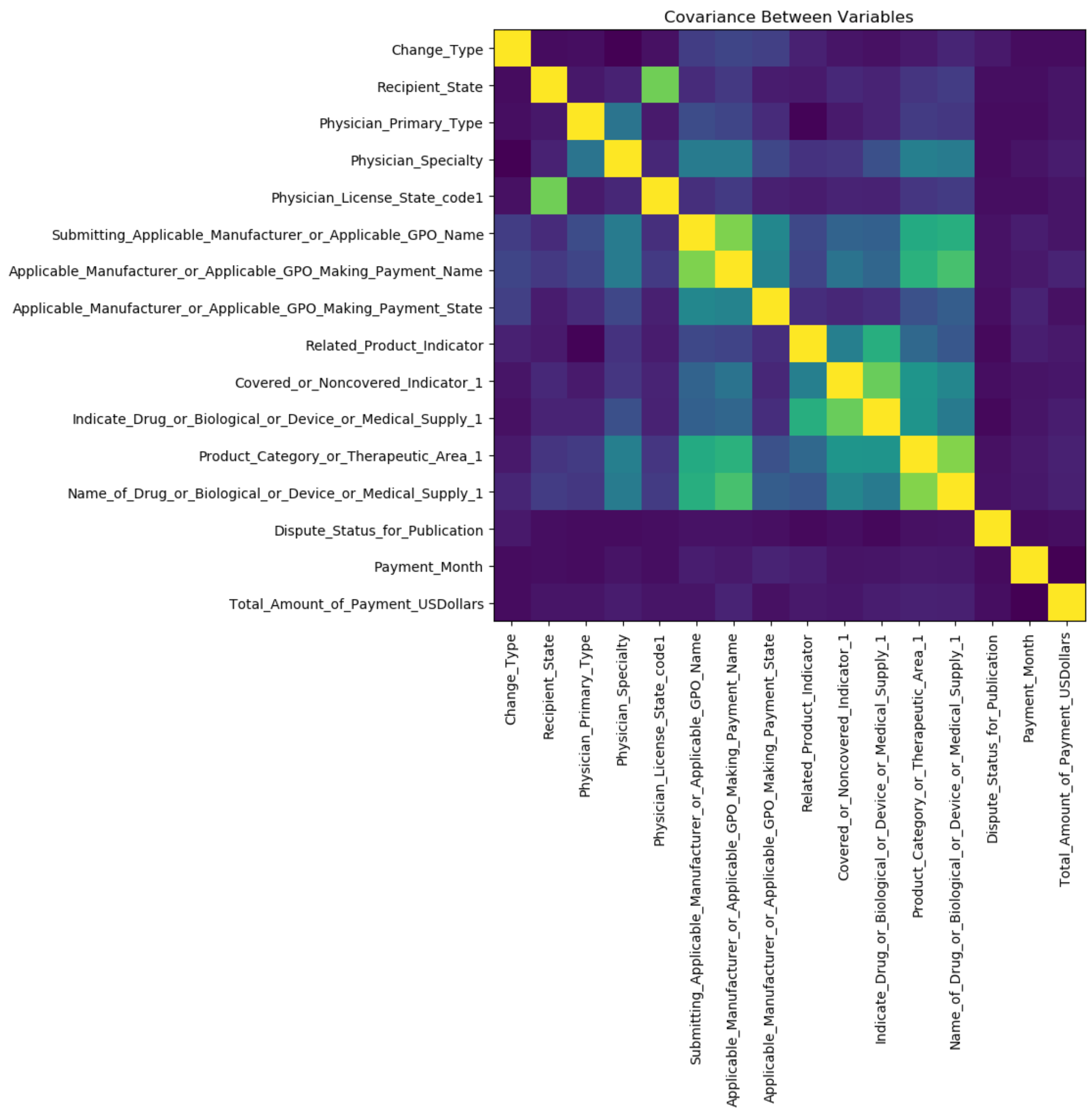
In [107]:

```
#Defining a function to replace the null values with NA
def null_categories_q3(Xa):
    return Xa.replace({np.nan:'NA'})
```

In [110]:

```python
from sklearn.preprocessing import StandardScaler
from category_encoders.target_encoder import *
#Checking the Correlation of other variables

#Creating a ColumnTransformer
preprocess_nan = ColumnTransformer(
    [('func', FunctionTransformer(func=null_categories_q3, validate=False), cat_
    ('scaler', StandardScaler(), cont_features)], remainder='passthrough')

#Transforming through the pipeline
pipeline_q3 = make_pipeline(preprocess_nan, TargetEncoder(impute_missing=False)
pipeline_q3.fit_transform(X, y.values)

#Scaling the Target Encoded Variables
from sklearn.preprocessing import scale
X_transformed = pipeline_q3.fit_transform(X, y.values)
X_scaled = scale(X_transformed)
cov = np.cov(X_scaled, rowvar=False)

#Plotting the Covariance Plot
plt.figure(figsize=(8, 8), dpi=100)
plt.imshow(cov)
plt.xticks(range(X.shape[1]), cat_features+cont_features, rotation=90)
plt.yticks(range(X.shape[1]), cat_features+cont_features)
plt.title('Covariance Between Variables');
```

```
/Users/Sarang/anaconda3/envs/coms007/lib/python3.6/site-packages/ipyke
rnel_launcher.py:17: DataConversionWarning: Data with input dtype int6
4, float64 were all converted to float64 by the scale function.
```



As we can see from above, there are some evident correlations:

- 'Physician_License_State_code1' is highly correlated with 'Recipient_State'
- 'Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name' is highly correlated with 'Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name'
- 'Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1' is highly correlated with 'Product_Category_or_Therapeutic_Area_1'

To tackle this, we will remove the correlated feature that has less number of null values. In case of a tie, we will remove the features that has more unique values to better generalize the data.

In [111]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19888 entries, 0 to 19887
Data columns (total 16 columns):
Change_Type                                                       1988
8 non-null object
Recipient_State                                                   1986
2 non-null object
Physician_Primary_Type                                            1977
0 non-null object
Physician_Specialty                                               1974
3 non-null object
Physician_License_State_code1                                     1977
0 non-null object
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name         1988
8 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name     1988
8 non-null object
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State    1886
7 non-null object
Related_Product_Indicator                                         1988
8 non-null object
Covered_or_Noncovered_Indicator_1                                 1808
5 non-null object
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1         1628
4 non-null object
Product_Category_or_Therapeutic_Area_1                            1577
0 non-null object
Name_of_Drug_or_Biological_or_Device_or_Medical_Supply_1          1621
6 non-null object
Total_Amount_of_Payment_USDollars                                 1988
8 non-null float64
Dispute_Status_for_Publication                                    1988
8 non-null object
Payment_Month                                                     1988
8 non-null object
dtypes: float64(1), object(15)
memory usage: 2.4+ MB
```

In [112]:

```
X.drop(columns=['Physician_License_State_code1',
               'Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_Name',

#Updating the Categorical Features
cat_features = list(X.columns)
cat_features.remove('Total_Amount_of_Payment_USDollars')
```

In [113]:

```
# Splitting the data in training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [114]:

```
#Creating a column transformer with null_categories_q3 function
preprocess_nan = ColumnTransformer(
    [('func', FunctionTransformer(func=null_categories_q3, validate=False), cat_
    ('scaler', StandardScaler(), cont_features)], remainder='passthrough')
```

In [115]:

```
#Creating Pipeline to standardize continuous features and target encode categor
pipeline_q3 = make_pipeline(preprocess_nan, TargetEncoder(impute_missing=False))
X_te = pipeline_q3.fit_transform(X_train, y_train.values)

#Running a 5-fold cross validation on the baseline model - Logistic Regression.
logreg_scores = cross_val_score(LogisticRegression(), X_te, y_train, cv=5)
print('Logistic Regression - Accuracy (Target Based): ', np.mean(logreg_scores))
```

```
Logistic Regression - Accuracy (Target Based):  0.8913911803460562
```

In [116]:

```
#Calculating the precision and recall
precision_q3 =  cross_val_score(LogisticRegression(), X_te, y_train, cv=5, scor
recall_q3 = cross_val_score(LogisticRegression(), X_te, y_train, cv=5, scoring=
print('Logistic Regression - Precision (Target Based): ', np.mean(precision_q3))
print('Logistic Regression - Recall (Target Based): ', np.mean(recall_q3))
```

```
Logistic Regression - Precision (Target Based):  0.897625720911762
Logistic Regression - Recall (Target Based):  0.8831265300374792
```

It looks like the Accuracy, Precision and Recall improve significantly. Let us now one-hot-encode the categorical features instead of target encoding and see how the results are affected.

In [117]:

```
#Creating Column Transformer for one-hot-encoding
preprocess_nan = ColumnTransformer(
    [('func', FunctionTransformer(func=null_categories_q3, validate=False), cat_
    ('scaler', StandardScaler(), cont_features)], remainder='passthrough')

preprocess_baseline_ohe = ColumnTransformer([('ohe', OneHotEncoder(handle_unkno
                                            np.arange(0,len(X_train.columns)-1)
```

In [118]:

```
#Creating pipeline for one-hot-encoding
pipeline_q3_1 = make_pipeline(preprocess_nan, preprocess_baseline_ohe, Logistic
logreg_scores = cross_val_score(pipeline_q3_1, X_train, y_train, cv=5)
print('Logistic Regression - Accuracy (OHE): ', np.mean(logreg_scores))
```

```
Logistic Regression - Accuracy (OHE):  0.9027888331059764
```

In [119]:

```
#Calculating the precision and recall
precision_q3_1 =  cross_val_score(pipeline_q3_1, X_train, y_train, cv=5, scorin
recall_q3_1 = cross_val_score(pipeline_q3_1, X_train, y_train, cv=5, scoring='r
print('Logistic Regression - Precision (OHE): ', np.mean(precision_q3_1))
print('Logistic Regression - Recall (OHE): ', np.mean(recall_q3_1))
```

```
Logistic Regression - Precision (OHE):  0.9145777401938282
Logistic Regression - Recall (OHE):  0.8882320710298461
```

It apprears that there isn't much difference in the way features are encoded. Both Target Based Encoding and One Hot Encoding give almost the same results. However, since there are a lot of categories, we will go ahead with using target encoding in the following.

Task 4

In this section, we will implement two classification models - Random Forest and Gradient Boosting. The preprocessing and feature engineering will be done as per the requirement of the model.

The dataset that would be used is the X_train and y_train that was created after we removed the features in the above tasks.

The following preprocessing and feature engineering steps would be done for both RandomForest and GradientBoosting. The steps are same as both of them are ensemble tree based models:

- Create a new category for the null values.
- Target encode the categorical variables.
- We will not scale the data as both are tree based model and scaling can be computationally expensive.

In [120]:

```
#Defining a function to replace the null values with NA
def null_categories_q4(Xa):
    return Xa.replace({np.nan:'NA'})

#Creating a column transformer with null_categories_q3 function
preprocess_nan_q4 = ColumnTransformer(
    [('func', FunctionTransformer(func=null_categories_q4, validate=False), cat_
```

First, we will implement the **Random Forest Classifier**. The parameters that we would be tuning are - n_estimators and max_depth over 3 values each.

In [121]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier

scoring = {'auc': 'roc_auc', 'accuracy':'accuracy', 'average_precision': 'avera
           'precision':'precision', 'recall':'recall'}

#Creating Pipeline for Random Forest
pipeline_rf = Pipeline(steps=[('nan',preprocess_nan_q4), ('te', TargetEncoder()
                             ('rf', RandomForestClassifier())])

#Defining the parameters over which grid-search would run: Note only two parame
#the model is computationally expensive - it makes the kernel crash on our mach

parameters = {'rf__n_estimators':[100, 500, 1000], 'rf__max_depth':[None, 3, 7]

#Running Grid Search
gcv_rf = GridSearchCV(pipeline_rf, param_grid= parameters, cv=3, scoring=scorin
gcv_rf.fit(X_train, y_train.values)

print("RF best parameters: {}".format(gcv_rf.best_params_))
```

```
RF best parameters: {'rf__max_depth': None, 'rf__n_estimators': 500}
```

In [122]:

```
print("accuracy RF: {}".format(np.mean(gcv_rf.cv_results_['mean_test_accuracy']
print("average_precision RF: {}".format(np.mean(gcv_rf.cv_results_['mean_test_a
print("precision RF: {}".format(np.mean(gcv_rf.cv_results_['mean_test_precision
print("recall RF: {}".format(np.mean(gcv_rf.cv_results_['mean_test_recall'])))
```

```
accuracy RF: 0.9403921218080511
average_precision RF: 0.979491694038916
precision RF: 0.9270312094666393
recall RF: 0.9559226800726699
```

Now, we will implement a **Gradient Boosting Classifier**. We would be tuning learning rate over 3 values (note that fewer hypterparameters are tuned due to computational limitations.

In [123]:

```
from sklearn.ensemble import GradientBoostingClassifier

#Creating Pipeline for Gradient Boosting
pipeline_gb = Pipeline(steps=[('nan',preprocess_nan_q4), ('te', TargetEncoder()),
                             ('gb', GradientBoostingClassifier())])

#Defining the parameters over which grid-search would run
parameters = {'gb__learning_rate':[.2, .1, .05]}

#Running Grid Search
gcv_gb = GridSearchCV(pipeline_gb, param_grid= parameters, cv=3, scoring=scoring,
gcv_gb.fit(X_train, y_train.values)

print("GB best parameters: {}".format(gcv_gb.best_params_))
```

```
GB best parameters: {'gb__learning_rate': 0.2}
```

In [124]:

```
print("accuracy GB: {}".format(np.mean(gcv_gb.cv_results_['mean_test_accuracy']
print("average_precision GB: {}".format(np.mean(gcv_gb.cv_results_['mean_test_a
print("precision GB: {}".format(np.mean(gcv_gb.cv_results_['mean_test_precision
print("recall GB: {}".format(np.mean(gcv_gb.cv_results_['mean_test_recall'])))
```

```
accuracy GB: 0.9434835076427998
average_precision GB: 0.9824335714538619
precision GB: 0.9321416412784284
recall GB: 0.9563851645505409
```

As we can see from the above results, both Random Forest and Gradient Boosting perform equally well. Note that the results that we get in this part are higher than what we got in Task 2 (Baseline) and improved Task 3 (Improved Baseline).

Task 5

In this section, we will identify features that are important to our best model. Since there is not much difference between Gradient Boosting and Random Forest as per Task 4, we will go ahead and choose Gradient Boosting to be the model that we will use in this task with the best parameter - learning_rate = 0.2

We will also study which features are most influential, and which features could be removed without decrease in performance.

In [125]:

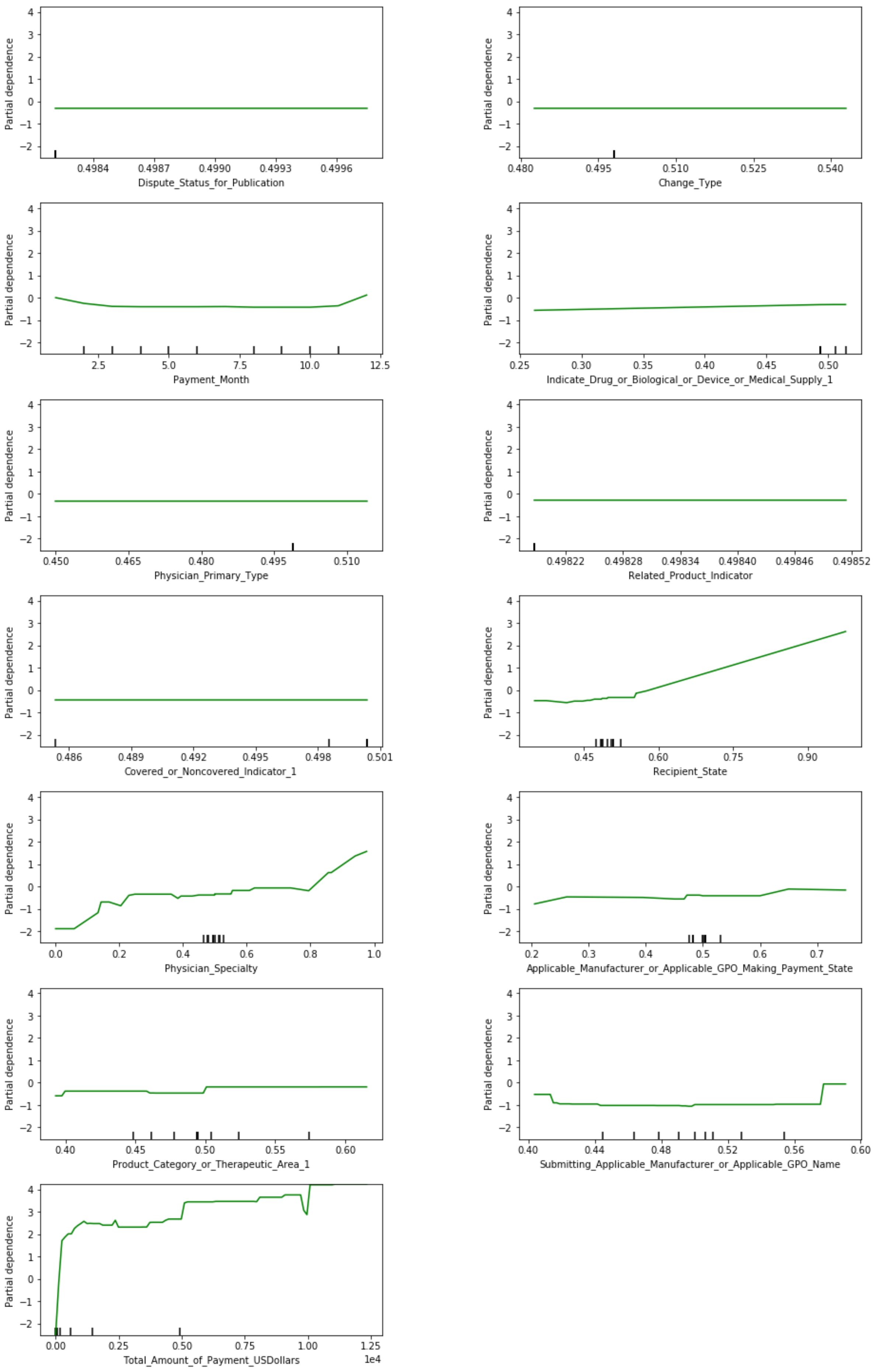
```
preprocess_nan_q5 = ColumnTransformer(
    [('func', FunctionTransformer(func=null_categories_q4, validate=False), cat_
```

First, we will plot the **Partial Dependence Curves** to see the dependence of each of the variable on the label individually.

In [155]:

```
from sklearn.ensemble.partial_dependence import plot_partial_dependence
x = preprocess_nan_q5.fit_transform(X_train)

_ = plt.figure(figsize=(100,100))

fig, axs = plot_partial_dependence(
    gcv_gb.best_estimator_.named_steps['gb'], TargetEncoder().fit_transform(x, 
    np.argsort(gcv_gb.best_estimator_.named_steps['gb'].feature_importances_)[-
    feature_names=cat_features+cont_features, n_jobs=-1)

plt.subplots_adjust(top=5, left=0, right=2)
```

```
<Figure size 7200x7200 with 0 Axes>
```


Partial dependence
Dispute_Status_for_Publication
Change_Type
Payment_Month
Indicate_Drug_or_Biological_or_Device_or_Medical_Supply_1
Physician_Primary_Type
Related_Product_Indicator
Covered_or_Noncovered_Indicator_1
Recipient_State
Physician_Specialty
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State
Product_Category_or_Therapeutic_Area_1
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name
Total_Amount_of_Payment_USDollars
1e4

The curves for 'Dispute_Status_for_Publication', 'Change_Type', 'Physician_Primary_Type', 'Related_Product_Indicator' and 'Covered_or_Noncovered_Indicator_1' are almost flat. This suggests that these features do not individually impact the target.

Now let us use **Recursive Feature Elimination**. This iterative model-based selection would help us drop features in a recursive fashion while takin into account impact of other features.

In [160]:

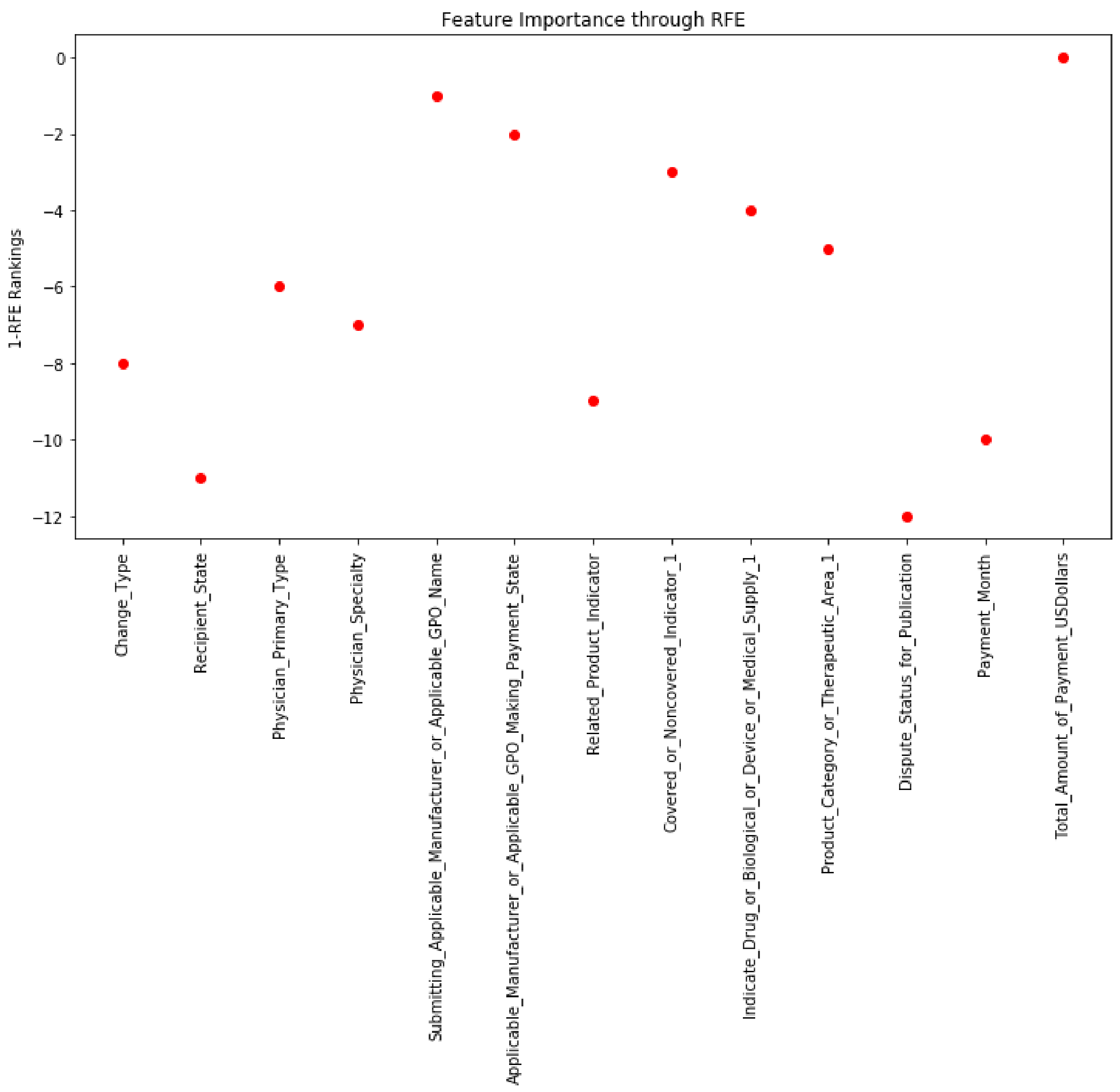
```
from sklearn.feature_selection import RFE

# create ranking among all features by selecting only one
rfe = RFE(gcv_gb.best_estimator_.named_steps['gb'], n_features_to_select=1)
rfe.fit(TargetEncoder().fit_transform(x, y_train), y_train)
rfe.ranking_
```

Out[160]:

```
array([ 9, 12,  7,  8,  2,  3, 10,  4,  5,  6, 13, 11,  1])
```

In [173]:

```python
#Plotting the importance of features obtained in RFE
fig = plt.figure(figsize=(12,6))
_ = plt.plot(1-rfe.ranking_, 'o', c='r')
_ = plt.ylabel("1-RFE Rankings")
_ = plt.xticks(range(X.shape[1]), cat_features+cont_features, rotation=90)
_ = plt.title('Feature Importance through RFE')
```



The results obtained in Partial Dependence Curve and Recursive Feature Elimination are slightly different. This might be due to the fact that Partial Dependence Curve ignore all the other features while reporting the importance whereas RFE adopts a iterative approach which takes into account the effect of other features.

Next, we will employ **Recursive Feature Elimination Cross Validation (RFECV)** that allows you to us efficient grid search for the number of features to keep.

In [189]:

```python
#Performing Recursive Feature Elimination Cross Validation
from sklearn.feature_selection import RFECV

X_train_transformed = TargetEncoder().fit_transform(x, y_train)

#Running RFECV to get the best parameters
rfe = RFECV(gcv_gb.best_estimator_.named_steps['gb'], cv=5)
rfe.fit(X_train_transformed, y_train)
print(rfe.support_)
print(list(np.array(cat_features+cont_features)[rfe.support_]))
```

```
[ True False  True  True  True  True  True  True  True  True False  Tr
ue
  True]
['Change_Type', 'Physician_Primary_Type', 'Physician_Specialty', 'Subm
itting_Applicable_Manufacturer_or_Applicable_GPO_Name', 'Applicable_Ma
nufacturer_or_Applicable_GPO_Making_Payment_State', 'Related_Product_I
ndicator', 'Covered_or_Noncovered_Indicator_1', 'Indicate_Drug_or_Biol
ogical_or_Device_or_Medical_Supply_1', 'Product_Category_or_Therapeuti
c_Area_1', 'Payment_Month', 'Total_Amount_of_Payment_USDollars']
```

As per RFECV, the above features are useful for the prediction. Now, we will build a model based on these features to see if the model performance improves.

In [192]:

```python
#Building Gradient Boosting Model based on the features selected by RFECV
from sklearn.model_selection import cross_validate

pipe_rfe_gb = make_pipeline(RFECV(gcv_gb.best_estimator_.named_steps['gb'], cv=5), gc
pipe_rfe_gb_scores = cross_validate(pipe_rfe_gb, X_train_transformed, y_train, cv=5,
```

In [195]:

```python
#Calculating the results
print("accuracy GB-RFECV: {}".format(np.mean(pipe_rfe_gb_scores['test_accuracy']
print("average_precision GB-RFECV: {}".format(np.mean(pipe_rfe_gb_scores['test_
print("precision GB-RFECV: {}".format(np.mean(pipe_rfe_gb_scores['test_precisio
print("recall GB-RFECV: {}".format(np.mean(pipe_rfe_gb_scores['test_recall'])))
print("auc GB-RFECV: {}".format(np.mean(pipe_rfe_gb_scores['test_auc'])))
```

```
accuracy GB-RFECV: 0.9410030811328248
average_precision GB-RFECV: 0.9806453397544246
precision GB-RFECV: 0.927698306795922
recall GB-RFECV: 0.9563411252410148
auc GB-RFECV: 0.9833411665443841
```

As we can see from the above, there is not much difference from the Gradient Boosting Model that was created in Task 4. This is probably due to the fact that the performance of the model in Task 4 was already very good. Hence, there might be a cieling effect limiting how much better scores we can obtain.

Task 6

In this task, we will create an 'explainable' model that will try to achieve the same performance results as our best model - Gradient Boosting built in the previous sections.

For this task, we will develop a simple tree. We will perform grid search to tune our parameters, use feature selection to reduce the number of features and also perform preprocessing suitable for Decision Tree. Since this model will be an explainable one, it will have small number of leaves/less depth.

We will focus on explaining the model globally i.e. we will identify the features that are important to the model.

In [238]:

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
pipeline_tree = Pipeline(steps=[('nan',preprocess_nan_q4), ('te', TargetEncoder()),

#Defining the parameters over which grid-search would run: Note only two parameters
#the model is computationally expensive - it makes the kernel crash on our machines

parameters = {'tree__max_depth': [None, 4, 10]}

#Running Grid Search
gcv_tree = GridSearchCV(pipeline_tree, param_grid= parameters, cv=3, scoring=scoring
gcv_tree.fit(X_train, y_train.values)

print("Tree best parameters: {}".format(gcv_tree.best_params_))
```

```
Tree best parameters: {'tree__max_depth': 4}
```

In [360]:

```
print("accuracy tree: {}".format(np.mean(gcv_tree.cv_results_['mean_test_accura
print("average_precision tree: {}".format(np.mean(gcv_tree.cv_results_['mean_te
print("precision tree: {}".format(np.mean(gcv_tree.cv_results_['mean_test_preci
print("recall tree: {}".format(np.mean(gcv_tree.cv_results_['mean_test_recall']
print("auc tree: {}".format(np.mean(gcv_tree.cv_results_['mean_test_auc'])))
```

```
accuracy tree: 0.9332037186019487
average_precision tree: 0.9214610912652677
precision tree: 0.9245028487325908
recall tree: 0.9433996366227194
auc tree: 0.9468890628021424
```

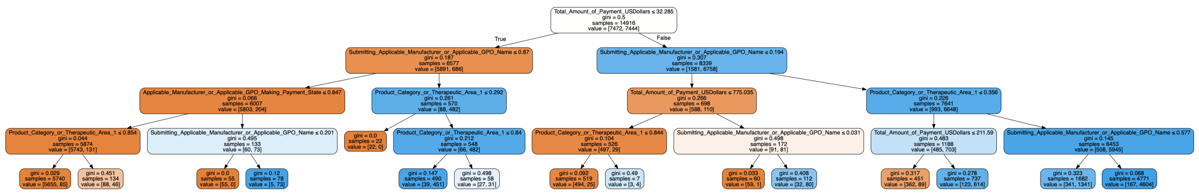
Note from the above that the performance of a single decision tree is nearly as good as best Gradient Boosting Model.

Now let us visulize the decision tree with the best parameters.

In [275]:

```python
#Visualizing the Decision Tree:
#source: https://medium.com/@rnbrown/creating-and-visualizing-decision-trees-wi

from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydot
import graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(gcv_tree.best_estimator_.named_steps['tree'], out_file=dot_data
                filled=True, rounded=True,
                special_characters=True, feature_names=(cat_features+cont_featu
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png(),width=2000, height=2000)
```

Out[275]:


Total_Amount_of_Payment_USDollars ≤ 32.285 gini = 0.5 samples = 14916 value = [7472, 7444]
True
False
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.87 gini = 0.187 samples = 6577 value = [5891, 686]
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.194 gini = 0.307 samples = 8339 value = [1581, 6758]
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State ≤ 0.847 gini = 0.066 samples = 6007 value = [5803, 204]
Product_Category_or_Therapeutic_Area_1 ≤ 0.292 gini = 0.261 samples = 570 value = [88, 482]
Total_Amount_of_Payment_USDollars ≤ 775.035 gini = 0.266 samples = 698 value = [588, 110]
Product_Category_or_Therapeutic_Area_1 ≤ 0.356 gini = 0.226 samples = 7641 value = [993, 6648]
Product_Category_or_Therapeutic_Area_1 ≤ 0.854 gini = 0.044 samples = 5874 value = [5743, 131]
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.201 gini = 0.495 samples = 133 value = [60, 73]
gini = 0.0 samples = 22 value = [22, 0]
Product_Category_or_Therapeutic_Area_1 ≤ 0.84 gini = 0.212 samples = 548 value = [66, 482]
Product_Category_or_Therapeutic_Area_1 ≤ 0.844 gini = 0.104 samples = 526 value = [497, 29]
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.031 gini = 0.498 samples = 172 value = [91, 81]
Total_Amount_of_Payment_USDollars ≤ 211.59 gini = 0.483 samples = 1188 value = [485, 703]
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.577 gini = 0.145 samples = 6453 value = [508, 5945]
gini = 0.029 samples = 5740 value = [5655, 85]
gini = 0.451 samples = 134 value = [88, 46]
gini = 0.0 samples = 55 value = [55, 0]
gini = 0.12 samples = 78 value = [5, 73]
gini = 0.147 samples = 490 value = [39, 451]
gini = 0.498 samples = 58 value = [27, 31]
gini = 0.092 samples = 519 value = [494, 25]
gini = 0.49 samples = 7 value = [3, 4]
gini = 0.033 samples = 60 value = [59, 1]
gini = 0.408 samples = 112 value = [32, 80]
gini = 0.317 samples = 451 value = [362, 89]
gini = 0.278 samples = 737 value = [123, 614]
gini = 0.323 samples = 1682 value = [341, 1341]
gini = 0.068 samples = 4771 value = [167, 4604]

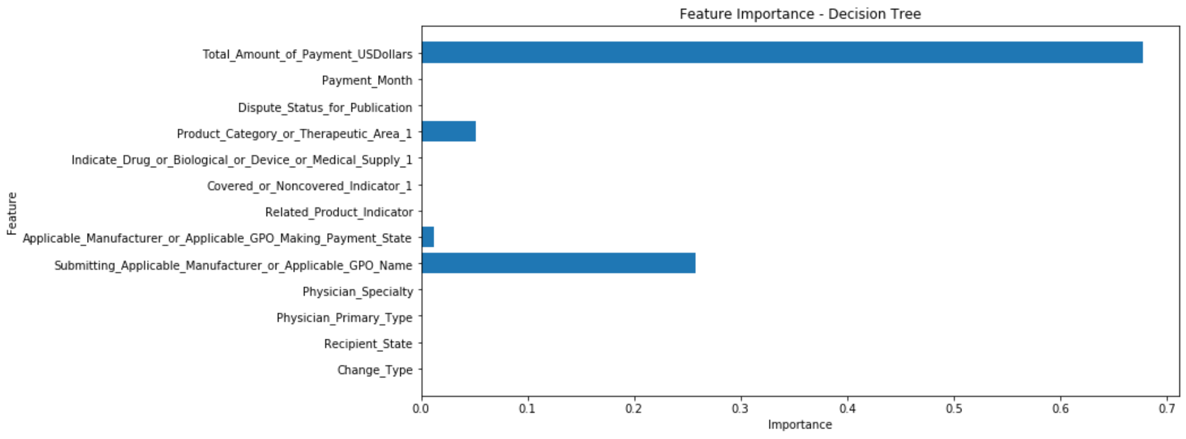
As we can see from the above tree, the model is very explainable. We can see that the tree is split on different features alongside the Gini coefficient allowing us to see which coefficient leads to maximum reduction in the Gini coefficient.

PS: Please ignore the small size of the chart due to large feature names, affecting the aspect ratio

Now, let us plot the Feature Importances

In [292]:

```python
#Plotting Feature Importances.
_ = plt.figure(figsize=(12,6))
_ = plt.barh(range(13), gcv_tree.best_estimator_.named_steps['tree'].feature_im
_ = plt.yticks(range(13), (cat_features+cont_features))
_ = plt.title('Feature Importance - Decision Tree')
_ = plt.xlabel('Importance')
_ = plt.ylabel('Feature')
```



Now we will build even a simpler model, where we will take only the 4 features as per the above chart that are important. We will eliminate all the other features:

- 'Total_Amount_of_Payment_USDollars'
- 'Product_Category_or_Therapeutic_Area_1'
- 'Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State
- 'Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name'

In [370]:

```
#Creating reduce training set

X_train_reduced = X_train[['Total_Amount_of_Payment_USDollars', 'Product_Catego
'Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State', 'Submitting_A

X_train_reduced.reset_index(inplace=True)
X_train_reduced.drop(columns='index', inplace=True)
```

In [372]:

```
preprocess_nan_q6 = ColumnTransformer(
    [('func', FunctionTransformer(func=null_categories_q4, validate=False), X_train_

pipeline_tree_update = Pipeline(steps=[('nan',preprocess_nan_q6), ('te', TargetEnco

#Defining the parameters over which grid-search would run: Note only two parameters
#the model is computationally expensive - it makes the kernel crash on our machines

parameters = {'tree__max_depth': [2, 3, 4]}

# # #Running Grid Search
gcv_tree_updated = GridSearchCV(pipeline_tree_update, param_grid= parameters, cv=3,
gcv_tree_updated.fit(X_train_reduced, y_train.values)

print("Tree best parameters: {}".format(gcv_tree_updated.best_params_))
```

```
Tree best parameters: {'tree__max_depth': 4}
```

In [373]:

```
print("accuracy tree (4 features): {}".format(np.mean(gcv_tree_updated.cv_resul
print("average_precision tree (4 features): {}".format(np.mean(gcv_tree_updated
print("precision tree (4 features): {}".format(np.mean(gcv_tree_updated.cv_resu
print("recall tree (4 features): {}".format(np.mean(gcv_tree_updated.cv_results_
print("auc tree (4 features): {}".format(np.mean(gcv_tree_updated.cv_results_['
```

```
accuracy tree (4 features): 0.9092026459283097
average_precision tree (4 features): 0.8957664351150787
precision tree (4 features): 0.8779306427984803
recall tree (4 features): 0.9515940559075123
auc tree (4 features): 0.9352025444704372
```

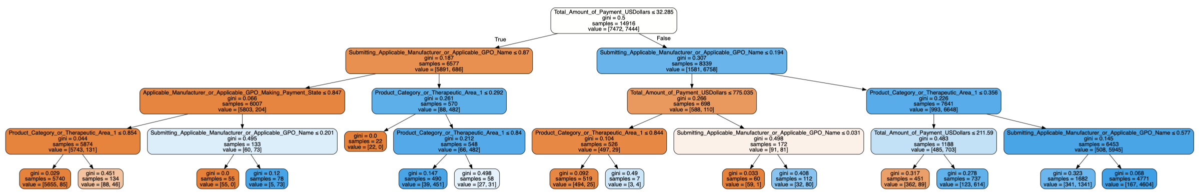
As we can see from the above, using only the 4 features does not decrease the performance to a huge extent. But at the same time we get a very good explainable model.

Let us now visualize this updated decision tree.

In [383]:

```python
#Visualizing the Decision Tree:
#source: https://medium.com/@rnbrown/creating-and-visualizing-decision-trees-wi

dot_data = StringIO()
export_graphviz(gcv_tree_updated.best_estimator_.named_steps['tree'], out_file=
                filled=True, rounded=True,
                special_characters=True, feature_names=X_train_reduced.columns)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png(),width=2000, height=2000)
```

Out[383]:


Total_Amount_of_Payment_USDollars ≤ 32.285 gini = 0.5 samples = 14916 value = [7472, 7444]
True
False
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.87 gini = 0.187 samples = 6577 value = [5891, 686]
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.194 gini = 0.307 samples = 8339 value = [1581, 6758]
Applicable_Manufacturer_or_Applicable_GPO_Making_Payment_State ≤ 0.847 gini = 0.066 samples = 6007 value = [5803, 204]
Product_Category_or_Therapeutic_Area_1 ≤ 0.292 gini = 0.261 samples = 570 value = [88, 482]
Total_Amount_of_Payment_USDollars ≤ 775.035 gini = 0.266 samples = 698 value = [588, 110]
Product_Category_or_Therapeutic_Area_1 ≤ 0.356 gini = 0.226 samples = 7641 value = [993, 6648]
Product_Category_or_Therapeutic_Area_1 ≤ 0.854 gini = 0.044 samples = 5874 value = [5743, 131]
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.201 gini = 0.495 samples = 133 value = [60, 73]
gini = 0.0 samples = 22 value = [22, 0]
Product_Category_or_Therapeutic_Area_1 ≤ 0.84 gini = 0.212 samples = 548 value = [66, 482]
Product_Category_or_Therapeutic_Area_1 ≤ 0.844 gini = 0.104 samples = 526 value = [497, 29]
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.031 gini = 0.498 samples = 172 value = [91, 81]
Total_Amount_of_Payment_USDollars ≤ 211.59 gini = 0.483 samples = 1188 value = [485, 703]
Submitting_Applicable_Manufacturer_or_Applicable_GPO_Name ≤ 0.577 gini = 0.145 samples = 6453 value = [508, 5945]
gini = 0.029 samples = 5740 value = [5655, 85]
gini = 0.451 samples = 134 value = [88, 46]
gini = 0.0 samples = 55 value = [55, 0]
gini = 0.12 samples = 78 value = [5, 73]
gini = 0.147 samples = 490 value = [39, 451]
gini = 0.498 samples = 58 value = [27, 31]
gini = 0.092 samples = 519 value = [494, 25]
gini = 0.49 samples = 7 value = [3, 4]
gini = 0.033 samples = 60 value = [59, 1]
gini = 0.408 samples = 112 value = [32, 80]
gini = 0.317 samples = 451 value = [362, 89]
gini = 0.278 samples = 737 value = [123, 614]
gini = 0.323 samples = 1682 value = [341, 1341]
gini = 0.068 samples = 4771 value = [167, 4604]